**Bildungszentrum Zürichsee**
Abteilung Informatik / Technik

# Codingstandards am BZZ

## Java Sourcecode

*„The first thing I would say is that when you write a program, think of it primarily as a work of literature. You're trying to write something that human beings are going to read. Don't think of it primarily as something a computer is going to follow. The more effective you are at making your program readable, the more effective it's going to be: You'll understand it today, you'll understand it next week, and your successors who are going to maintain and modify it will understand it. “ Donald Knuth*

## Inhaltsverzeichnis



**Symbole**


Zwingende Vorschriften


Empfehlungen


Begründete Ausnahmen

# 1 Bezeichner

*Bezeichner sind die «Namen» von Klassen, Methoden, Attributen, Variablen und Konstanten.*

## 1.1 Sprechende Bezeichner

*Ein sprechender Bezeichner sagt etwas über den Sinn und Zweck einer Komponente aus. Durch den Einsatz von sprechenden Bezeichnern werden viele Kommentare überflüssig.*

Die Bezeichner (Namen) von Klassen, Methoden, Attributen, Variablen und Konstanten müssen sprechend sein.

Schleifenzähler innerhalb einer kurzen Iteration (max. 5 Zeilen) dürfen auch kürzere Variablennamen wie **i**, **j** verwenden.

**Beispiel**

```
public class category {

    String[] categories = {"Getränke", "Vorspeisen", "Hauptgang", "Dessert"};
    public int readCategoryByTitle(String categoryTitle) {
        int categoryId = -1;

        for (int i=0; i < categories.length; i++) {
            if (categories[i].equals(categoryTitle)) {
                categoryId = i;
            }
        }

        return categoryId;
    }
}
```

## 1.2 Schreibweise

*Durch eine einheitliche Schreibweise verbessern Sie die Lesbarkeit Ihres Sourcecodes.*


Klassennamen bestehen aus einem Substantiv und beginnen mit einem Grossbuchstaben (z.B. Account).


Die Bezeichner von Methoden, Attributen und Variablen beginnen mit einem Kleinbuchstaben.


Konstanten werden nur mit Grossbuchstaben geschrieben.


Zusammengesetzte Bezeichner sollten in camelCase geschrieben werden; z.B. getFirstname().


Vermeide Sonderzeichen, Leerzeichen und Umlaute in allen Bezeichnern, sowie in Datei- und Ordnernamen.

**Beispiel**

```
public class Project {
    private String projectTitle;
    private static final int NUMBER = 10;

    /**
     * @param projectTitle
     *            the projectTitle to set
     */
    public void setProjectTitle(String projectTitle) {
        this.projectTitle = projectTitle;
    }

}
```

# 2 Sichtbarkeit von Attributen

## 2.1 Datenkapselung

*Der Einsatz von* **public***-Attributen untergräbt das Prinzip der Datenkapselung.*

Die Attribute einer Klasse werden entweder **private** (Normalfall) oder **protected** (Vererbung) definiert.

## 2.2 Zugriff auf Attribute

*Die konsequente Verwendung von getter/setter-Methoden unterstützt* **lazy initialization** *und vermeidet Konflikte mit lokalen Variablen.*

Der Zugriff auf Attribute soll auch in derselben Klasse mittels Getter/Setter-Methoden erfolgen.

**Beispiel**

```
public class Project {
    private String projectTitle;
    private Category category;
    private static final int NUMBER = 10;

    /**
      * default constructor
      */
    public Project() {
        setProjectTitle(“”);
        setCategory(new Category());
    }

    /**
     * @param projectTitle
     *            the projectTitle to set
     */
    public void setProjectTitle(String projectTitle) {
        this.projectTitle = projectTitle;
    }

    /**
      * @param category
      *           the category to set
      */
    public void setCategory(Category category) {
        this.category = category;
    }
}
```

# 3 Darstellung und Formatierung

*Eine klare und einheitliche Programmierung erleichtert das Lesen eines Sourcecodes.*

## 3.1 Programmblöcke

*Ein Programmblock wird zwischen geschweiften Klammern `{ … }` eingefasst.*


Die Positionierung der geschweiften Klammern ist einheitlich innerhalb des ganzen Projekts.


Innerhalb eines Programmblocks werden die Zeilen um 2 oder 4 Stellen eingerückt. Die Einrückung ist im ganzen Sourcecode einheitlich.

**Beispiel: Variante 1 (K&R style)**

```
    if (anredeCode == 1) {
        return "Herr";
    } else {
        return "Frau";
    }

    try {
         ...
    } catch (...) {
        ...
    }
```

**Beispiel: Variante 2**

```
    if (anredeCode == 1)
    {
        return "Herr";
    }
    else
    {
        return "Frau";
    }

    try
    {
         ...
    }
    catch (...)
    {
        ...
    }
```

## 3.2 Codezeilen

*Eine Codezeile sollte immer auf einen Blick erfasst werden können.*

Eine Codezeile ist maximal 80 (oder 120) Zeichen lang. Längere Zeilen werden umgebrochen.

Ordnen Sie gleiche Elemente wie Bezeichner oder Bedingungen untereinander an.

**Beispiel**

```
if (
     ( Integer.parseInt(eingabeMenge) >= mindestMenge  &&
       Integer.parseInt(eingabeMenge) <= maximalMenge
     ) ||
     ( kunde.kundenArt.equals("Stammkunde") &&
       Integer.parseInt(eingabeMenge) > 0
     )
   ) {
    ....
}
```

## 3.3 Konstruktoren

*Konstruktoren versetzen ein neu erzeugtes Objekt in einen definierten Anfangszustand.*

Die Konstruktor-Methode(n) stehen an erster Stelle, um die Übersichtlichkeit zu erhöhen.

Die Initialisierung von Attributen erfolgt im Konstruktor, nicht bei der Deklaration.

**Beispiel**

```
public class Project {
    private String projectTitle;
    private Category category;

    /**
      * default constructor
      */
    public Project() {
        setProjectTitle(“”);
        setCategory(new Category());
    }

...
}
```

## 3.4 Anordnung der Methoden

*Durch eine geeignete Reihenfolge fällt es leichter, die richtige Methode effizient zu finden.*

Ähnliche Methoden sollen immer in der gleichen Reihenfolge angeordnet werden.

**Beispiel**

1. Konstruktoren
   1.1. Default constructor
   1.2. Alternative Konstruktoren
2. Methoden die eine Verarbeitung auslösen (z.B. savePerson())
3. Methoden die auf ein Ereignis reagieren (Listener, Events)
4. Getter/Setter-Methoden
   4.1. Getter/Setter mit integrierter Logik
   4.2. «Normale» Getter/Setter die Attribute 1 zu 1 lesen oder schreiben.

# 4 Kommentare und Dokumentation

*Zur Dokumentation des Sourcecodes verwenden wir Javadoc-Kommentare. Dadurch ersparen wir uns eine separate Beschreibung der Klassen und Methoden in einem Textdokument.*

## 4.1 Klassenkopf

*Der Klassenkopf ist die Visitenkarte einer Klasse. Er informiert den Programmierer über Aufgabe und Version dieser Klasse.*

Jede Klasse hat einen Javadoc-Klassenkopf. Dieser enthält mindestens die Angaben zu:

- Kurzbeschreibung
- @author: Autor(en)

Sofern Sie keine Versionsverwaltung (z.B. git) verwenden, sollten ausserdem

- @since: Letztes Änderungsdatum
- @version: Aktuelle Version des Sourcecodes

im Kopf stehen.

**Beispiel**

```
/**
 * category to which a project is assigned
 *
 * @author Marcel Suter
 * @since 2017-09-19
 * @version 1.0
 *
 */
```

## 4.2 Methodenkopf

*Der Methodenkopf liefert alle Angaben über die Aufgabe und Schnittstelle einer Methode.*

Jede Methode hat einen Javadoc-Kopf mit den Angaben:

- Kurzbeschreibung
- @param: Beschreibung der Parameter (falls vorhanden)
- @return: Beschreibung des Returnwerts (falls vorhanden)
- @exception: Beschreibung der Exceptions (falls vorhanden)

**Beispiel**

```
/**
  * load a project from the database
  *
  * @param  projectId      the unique id of a project
  * @param  categoryTitle  the category title to be searched
  * @return enum Result
  * @throws Exception("duplicate entry")
  */
```

## 4.3 Kommentare

*Kommentare sind wie das Salz in der Suppe: Zuwenig und es schmeckt nicht, zu viel und es ist ungeniessbar.*

Jeder Programmblock, dessen Aufgabe nicht offensichtlich ist, wird in einem Kommentar beschrieben.

Ein kurzer Blockkommentar ist oftmals sinnvoller als Zeilenkommentare, die über den Bildschirm hinausreichen.

**Beispiel**

```
private Connector getConnector() {
    /* create a new connector, if not exists
    /* and establish connection if needed
    if (this.connector == null) {
        setConnector(new Connector());
    }
    if (this.connector.getHandle() == null) {
        this.connector.connect();
    }

    return this.connector;
}
```